

Moodle UE

Ivo Rego

Arquitectura de Sistemas e Computadores

II

Página principal ► Licenciaturas ► 2015/2016 - Semestre Ímpar ► INFO863 ► Práticas ► Sumários / Exercícios

NAVEGAÇÃO

Página principal

■ Painel do utilizador

Páginas do site

Disciplina atual

INFO863

Participantes

Medalhas

Geral

Teóricas

Práticas

■ Recursos

■ Sumários / Exercícios

■ Exercícios

1ª Aula

■ Exercícios
2ª Aula

■ Exercícios
3ª Aula

■ Exercícios
4ª Aula

■ Exercícios
5ª Aula

■ Exercícios
6ª Aula

Sumários das práticas

1ª Aula (2015/9/14)

Código MIPS: interpretação, análise, alteração e criação.

► Exercícios:

2ª Aula (2015/9/21)

Exercícios 1.5, 1.6. e 1.7 do COD5: análises de desempenho envolvendo o número de instruções executadas por segundo, o número de ciclos e o número de instruções para um programa, a frequência do relógio, o CPI, o tempo de CPU, distribuições de instruções e a influência do compilador.

Apresentação do exercício 1.9 do COD5: um cheiro da análise de desempenho em multiprocessadores.

Outros exercícios recomendados (COD5): 1.11.1, 1.11.3, 1.11.4, 1.11.6 a 1.11.11.

3ª Aula (2015/9/28)

Exercícios 4.1 e 4.2 do COD5: operação da implementação MIPS da figura; valores dos sinais de controlo; actividade das unidades funcionais na execução de algumas instruções e implementação de novas instruções.

▼ Exercício (tempos de resposta)

6. Considere a implementação MIPS da figura e os seguintes tempos de resposta das unidades funcionais. (Assuma que toda a lógica não mencionada na tabela tem tempo de resposta 0.)

Mem	Add(PC+4)	Muxes	ALU	Banco de	Mem	Sign-
-----	-----------	-------	-----	----------	-----	-------

Exercícios**7ª Aula****Avaliação**

As minhas
disciplinas

instr.				registos	dados	Extend
200ps	70ps	20ps	90ps	80ps	250ps	15ps

Com estes tempos de resposta, qual o menor tempo que demora desde o instante em que o endereço está presente à saída do PC até ao valor final ter sido escrito no seu destino?

- a. Para a instrução `and`.
- b. Para a instrução `lw`.

ADMINISTRAÇÃO**O**

Administração
da disciplina

4ª Aula (2015/10/5)

Exercícios 4.3.1 e 4.3.2 (COD5) (sobre a implementação da Figura 4.17): cálculo do caminho crítico de uma instrução a partir da latência dos circuitos.

Exercício 4.7 (COD5): análise fina do funcionamento da implementação monociclo do MIPS (Figura 4.17).

▼ Exercício (alterações à implementação)

7. Pretende-se modificar a implementação monociclo do MIPS da Figura 4.17 para suportar a instrução `jr`.

- a. Quais das unidades funcionais existentes serão usadas pela instrução? (Não esqueça os multiplexers.)
- b. Que unidades funcionais adicionais é necessário acrescentar?
- c. Que novos sinais de controlo são necessários?
- d. Quais os valores de todos os sinais de controlo? (Em relação à ALU, indique só a operação que irá ser realizada.)

Outros exercícios recomendados: o exercício anterior com outras instruções MIPS (e.g., `addi`, `bne`, `jal`, `lui`, `movz`, etc.), e o 4.4.

5ª Aula (2015/10/12)

▼ Exercício (alterações à implementação monociclo)

7. (Continua da aula anterior)

Pretende-se modificar a implementação monociclo do MIPS da Figura 4.17 para suportar as instruções `bne` e `jal`.

- a. Quais das unidades funcionais existentes serão usadas pela instrução? (Não esqueça os multiplexers.)
- b. Que unidades funcionais adicionais é necessário acrescentar?
- c. Que novos sinais de controlo são necessários?
- d. Quais os valores de todos os sinais de controlo? (Em relação à ALU, indique só a operação que irá ser realizada.)

Exercícios 4.8.1 a 4.8.5 (COD5): latências das unidades funcionais; duração de um ciclo de relógio no pipeline MIPS.

▼ Exercícios (execução pipelined)

8. Calcule o CPI para a implementação *pipelined* do MIPS: para 1 instrução; para um milhão de instruções; e para um milhão de instruções quando 1 em cada 5 é atrasada um ciclo.
9. Para o código abaixo:
- Identifique todas as dependências (de dados) existentes. (Cuidado com os saltos.)

1.	or	\$3, \$0, \$0
2.	or	\$8, \$5, \$0
3.	início:	beq \$8, \$0, fim
4.		addiu \$4, \$4, -4
5.		lw \$9, 0(\$4)
6.		add \$3, \$3, \$9
7.		beq \$0, \$0, início
8.		addi \$8, \$8, -1
9.	fim:	sub \$3, \$7, \$3

Outros exercícios recomendados: 4.5, 4.9.2 a 4.9.6 (na alínea 4.9.6, onde está MEM deve ler-se WB).

6ª Aula (2015/10/19)

▼ Exercícios (execução *pipelined*)

9. (Continua da aula anterior)

Na resolução deste exercício, assuma que a decisão sobre se um salto condicional é ou não efectuado é tomada no andar EX e que é usado um *delay slot*.

Para o código abaixo:

1.	or	\$3, \$0, \$0
2.	or	\$8, \$5, \$0
3.	início:	beq \$8, \$0, fim
4.		addiu \$4, \$4, -4
5.		lw \$9, 0(\$4)
6.		add \$3, \$3, \$9
7.		beq \$0, \$0, início
8.		addi \$8, \$8, -1
9.	fim:	sub \$3, \$7, \$3

b. Identifique os conflitos de dados.

c. Introduza os *nop* necessários para eliminar os conflitos num pipeline sem *forwarding*.

- d. Introduza os nops necessários para eliminar conflitos no pipeline com *forwarding*.

e. Apresente a evolução do estado do pipeline com *forwarding* durante a execução do código, sendo a instrução 3 executada 2 vezes. Indique todos os atrasos introduzidos e todos os pontos onde foi necessário fazer *forwarding* de algum valor.

f. Considerando somente os ciclos 5 a 12 da simulação feita na alínea anterior, calcule a percentagem dos ciclos de relógio em que todos os andares do pipeline são efectivamente usados por alguma instrução.

g. Para o ciclo da execução em que o conteúdo do pipeline é o indicado abaixo, qual é o valor dos sinais de controlo usados em cada um dos andares?

Andar	Instrução
IF	add \$3, \$3, \$9
ID	lw \$9, 0(\$4)
EX	addiu \$4, \$4, -4
MEM	beq \$8, \$0, fim
WB	or \$8, \$5, \$0

h. Modifique o código de modo a que a sua execução se possa dar sem a introdução de qualquer atraso.

Exercício 4.10.1 (COD5): pipeline MIPS com uma única unidade de memória.

Outros exercícios recomendados: 4.11., 4.13, 4.14, 4.15, 4.16.1 a 4.16.3.

Última alteração: Quinta, 22 Outubro 2015, 18:50

Nome de utilizador: Ivo Rego. (Sair)

INF0863

02 ,02 ,82	no	1
02 ,22 ,82	no	2
m77 ,02 ,82	funcção: ped	3
4- ,42 ,42	rbbo	4
(4220 ,02	wj	5
82 ,62 ,82	bb0	6
funcção ,02 ,02	ped	7
1- ,82 ,82	bb0	8
62 ,52 ,82	due :m77	9

Aero. ④ Silvage - Circuito silencioso

Conselho mundo [Hz]

[Hz]

Procurar (dividir) e'
refinhar os dados de
controle e
que continuam

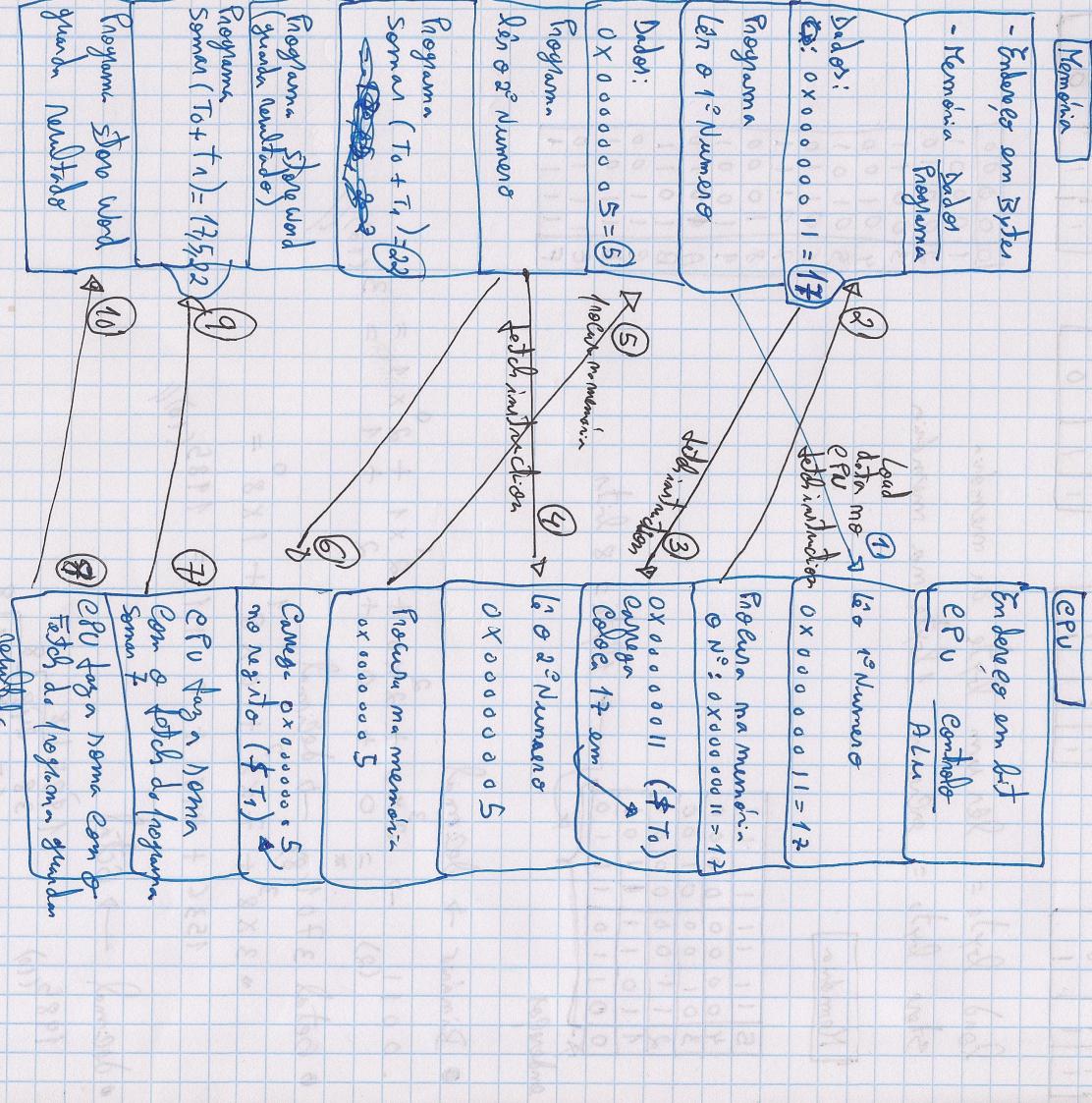
12.5 Hz

10 milhares o mais de instâncias / Seg.

Overclocking - ~~desempenho~~ desempenho

Giga - 10^9

memória relativa - o conteúdo é temporário
memória mao relativa - o conteúdo é permanente



Un programa es un conjunto de números en memoria que realizan una función o función (código Maquina)

AND

A	B	A \times B
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	A \oplus B
0	0	0
0	1	1
1	0	1
1	1	1

NOT

A	B	A \neg B
0	0	1
0	1	0
1	0	0
1	1	0

exclusive OR
XOR

A	B	A \otimes B
0	0	0
0	1	1
1	0	1
1	1	0

load byte = ler um byte da memória

store byte = escrever 1 byte na memória

Memória

5	11	11	11	11	11
4	00	00	00	00	00
3	01	00	01	00	
2	11	00	00	00	
1	10	11	11	01	
0	01	10	11	10	

endereços

1 byte = 8 bits

0	0	0	0
1	0	0	1
2	0	0	10
3	0	0	11
4	0	1	00
5	0	1	01
6	0	1	10
7	0	1	11
8	1	0	00
9	1	0	01
A	1	0	10
B	1	0	11
C	1	1	00
D	1	1	01
E	1	1	10
F	1	1	11

• Binário \rightarrow decimal

$$0011_{(2)} \rightarrow 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 1 = 3_{(10)}$$

• octal 3701₍₈₎ \rightarrow decimal

$$* 3 \times 8^3 + 7 \times 8^2 + 0 \times 8^1 + 1 \times 8^0 =$$

$$1536 + 448 + 0 + 1 = 1985_{(10)}$$

• decimal \rightarrow octal

$$\begin{array}{r} 1985 \\ \overline{38} \quad 248 \\ 65 \quad 08 \quad 31 \\ \hline 1 \quad 0 \quad 7 \quad 3 \end{array}$$

octal 3701₍₈₎

Instruções

- $SLT \rightarrow$ net ist lens then

$SLT \quad T_0, T_1, T_2 \Leftrightarrow \begin{cases} T_0 = 1 & \text{se } T_1 < T_2 \\ 0 & \text{se } T_1 \geq T_2 \end{cases}$

Exemplo: add $T_1, 3000, 2$

add $T_2, 3000, 3$

$SLT T_0, T_1, T_2$

- Branch operations: if then do codigos Programa

freq	T_0, T_1, XPT_0	if $T_0 = 2 T_1$
else	T_0, T_1, ABE	is not equal

[not] delayed branching

- zero $0x00000000$

• $a_0 - a_3 \rightarrow$ mudar para tratar argumentos para funções

• $b - v_4$ mudar para devolver resultado de numa função

• $T_0 \sim T_1$ mudar mas alterações

• $\# T_1 = 5 \times T_0$

$SLL T_1, T_0, 2$

add T_1, T_1, T_0

$T_1 = 15$

- # SR a - shift Right automóvel do bit_i

$T_1 = 65535 = 0xFFFF$

mudar de 32 bits

$SRa \quad T_0, T_1, 4$

$T_0 = 0 \times 0FFFF$

se $\Rightarrow MSB = 1$ entrem 1

se $\Rightarrow MSB = 0$ entrem 0

Ex: $0x000000FFFF$ com $SRa 4 =$
 $= 0000000FFFF_{16}$

SRL (Jumps)

$T_1 = 0xFFFF$

SRL $T_0, T_1, 4$

$T_0 = 0x FFFF_{(16)}$

$T_0 = 0xFFFFFF$

Reg $T_0, T_1, xRTO$

lme T_0, T_1, ABe

xRTO:

ABE;

Shift deslocamento orig. da dir.

SLL $T_0, T_1, \frac{3}{T_0}$ → shift left 3 bits, a partir o zero ficando

com o menor numero quando o zero muda
novo deslocamento entra em geral a esquerda

[Ex:]

0	1	0	...	1	1
---	---	---	-----	---	---

[Exemplo:] add: $T_1, 300, 2$

→

$T_0 \dots 110$

SLL $T_0, T_1, 4$

$\frac{3}{T_0}$ shift de 4 bits

O resultado T_2 ficando

0	0	1	0	0	0
---	---	---	---	---	---

entram 4 bits da esquerda

• Multíplicador de registrador:

$$1537 \times \underbrace{100}_{\sim^2} = 153700$$

$$100 = 2^{-2}8, 8, \boxed{128} \times \boxed{1537} = 43036$$

add: $T_1, 300, 1537$

srl $T_2, T_1, 7$

Sub: $T_2, T_2, 43036$

$T_2 = 153700$

resultado de $T_1 - 43036$

Exercise 2

$$13 \times 32 = 416$$

addi $T_1, 32, 13$

SLL $T_2, T_1, 5$

$$\boxed{T_2 = 416}$$

$$32 = 2^5$$

$$13 \times 33 = 429$$

addi $T_1, 32, 13$

SLL $T_2, T_1, 5$

add T_2, T_2, T_1

$$\boxed{\overline{T_2} = 429}$$

Exercise 3

$$33 = 2^5 + 1$$

Exercise 4

$$5 \times 8 = 40$$

$$2^3 = 8$$

$$101 \times 1000 = 101000$$

addi $T_1, 32, 13$

SLL $T_2, T_1, 5$

add T_2, T_2, T_1

$$\boxed{T_2 = 40}$$

[SRA] Shift Right Arithmetic SRA $T_0, T_0, 1$

$$1000 \dots 100$$

$$000 \dots 101$$

$$11000 \dots 10$$

$$0000 \dots 10$$

introducing zero on gno conforms or \neq MSB

(SRL)

shift Right logical return register divided by 2

register divide by 2

SRL $T_0, T_0, 2$

SRA $T_0, T_0, 2$

• [Obj] en el num. expresas Dígita

LUI ~~Load~~ Pa

Lui To, 0x1234

Example:

Add To, Zne, 0x1234

S&P To, To, 16

0x1234 [56+8]
R
16bit

Manda LUI para la 1 instrucción que crea una variable o programa

Código MIPS: interpretação, análise, alteração e criação.

▼ Exercícios

1. Dado o código seguinte, assumindo $\$a1 > 0$ e resultado em $\$v0$:

```

    lw      $v0, 0($a0)
ciclo: addi   $a1, $a1, -1
        beq    $a1, $0, fim
        nop
        addiu $a0, $a0, 4
        lw      $t0, 0($a0)
        slt   $t1, $v0, $t0
        beq   $t1, $0, ciclo
        nop
        or     $v0, $t0, $t0
        j      ciclo
        nop
fim:   jr    $ra
        nop
    
```

anotações:
 a. Início do bloco de código.
 b. Início da estrutura de repetição.
 c. Início da estrutura condicional.
 d. Início da estrutura condicional.

- 2.
- Escreva código MIPS para uma função que calcula $A = A + B$, onde A e B são duas matrizes $N \times N$.
 - Reescreva-o, de modo a aproveitar os *delay slots* ao máximo.

a. Argumentos da função:

$\$a0$ endereço de A $\$a1$ endereço de B $\$a20 \leq N \leq 45000$

- b. Quantas instruções serão executadas na função que escreveu se $N (\$a0)$ for 1000?

3. Argumentos da função:

$\$a0$ endereço do vetor $\$a1$ dimensão do vetor (≥ 0)

4. Escreva código MIPS para uma função que calcula o número de ocorrências do valor x num vector (≥ 0) de inteiros.

Argumentos da função:

$\$a0$ endereço do vetor $\$a1$ dimensão do vetor (≥ 0) $\$a2$ x

5. Escreva código MIPS para uma função que calcula o número de valores menores que x num vector (≥ 0) de inteiros, e o número de valores maiores que x nesse vector.

Argumentos da função:

$\$a0$ endereço do vector $\$a1$ dimensão do vector (≥ 0) $\$a2$ x

⑪ Out
Rendidas

• data
Array: .word 2,4,3,1,5 # cada word tem 4 bits

• addi a₁, \$zero, 5 # tamanho do array

lw v₀, 0(\$a₀) # retorna tamanho + 1 = 5 para v₀ em que o 1 indica final

Tenham Valores de Comparação

c-eq:

addi a₁, a₁, -1 # decrementa 1 word no tamanho do array

c-eq: # faz o mesmo em todos os elementos do array (-1)

addi a₁, a₁, -1 # decrementa 1 word no tamanho do array

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t1 # se a₁ for menor que a₀ far label t1

addi a₀, a₀, 4 # incrementa 4 bits da referência e endereço de

addi a₀, a₀, 4 # incrementa valor do array em referência e

beq a₁, a₀, \$t0 # se a₁ for menor que a₀ far label t0

• Invertendo palavras: - c-eq: em v₀ o endereço de 1º elemento do array que obtemos

- c-eq: decrementa \$a₁ que contém o comprimento do array

- implementa a₀ & comeca em T₀, o elemento do endereço \$a₀

• Compara o endereço v₀ com T₀, se $v_0 < T_0 \quad T_1 = 1$

$v_0 \geq T_0 \quad T_1 = 0$

• Se $v_0 \geq T_0$ o beq foge indo para "c-eq" executando normalmente

met (devido branching)

①

②

③

④

⑤

⑥

LW $V_0, 0(a_0)$ / Reduces 2
~~LW $T_0, 0(a_0)$~~ LW $V_0, 0(a_0)$ / Reduces 3
Reduces 3

edr:

addi $a_1, a_{11}, -1$
addi $a_0, a_0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

edr:

addi $a_1, 0, 4$
addi $a_0, a_0, 4$

②

$$1 + (1 \times 1000) + 2 + \frac{2}{\text{# instr}} = 9005 \text{ instrucciones ejecutadas}$$

①

instr

②

mem

③

RA

④

mem

⑤

mem

⑥

mem

return values

③

incrementar o '4' fin. nos words

~~lbg a1, \$0, \$im~~

lbg a1, \$0, \$im } Verifica se a condição (≥ 0) de incremento
no registro V0

lbg \$t0 0(a0)

addiu V0, V0, T0

addi a1, a1, -1 decrementa o contador

lbg a1, 0(celo)

addiu a0, a0, 4 # delay slot

fin:

j17 ra

not

④

lbg

lbg

celo: 07 V0, 0, 0

lbg a1, 0(a0) } fin

addi a1, a1, -1 # delay slot

lbg t0, 0(a0) } fin

lbg t0, a2, celo } fin

addiu a0, a0, 4 # delay slot

celo:

addiu V0, V0, 1 # delay slot

fin:

j17 ra

outros lugares:



otimizado

LW T0, 0(a0)

OR V0, a0, 0

celo:

addiu a1, a1, -1

lbg a1, 0, \$im

not

seq

T1, T0, a2

add V0, V0, T1

addi a0, a0, 4

lw t0, 0(a0)

addiu a1, a1, -1

not

fin:

j Ra

not

(5) OR $v_0, 0, 0$

OR

$v_1, 0, 0$

outra solução

OJO : $f_{reg} a_1, 0, t_{im}$

$a_1, a_1, -1$

$t_0, 0, (ao)$

t_0, t_0, a_2

t_1, t_0, a_2

$t_1, 0, (máx)$

$a_0, a_0, 4$

f_{echo}

$add: v_0, v_0, 1$

$min: v_0$

$sdt t_1, a_2, t_0$

$beg t_1, 0, echo$

$met #, desenhar máx intilizado$

f_{echo}

$add: v_1, v_1, 1$

met

f_{echo}

$add: v_0, v_0, 1$

met

tim : f_{reg} delay set não intilizado

lw $T_0, 0, (ao)$

$v_0, 0, 0$

$v_1, 0, 0$

$t_0, 0, 0$

$t_1, a_1, -1$

$a_1, a_1, -1$

$t_0, (ao)$

t_0, t_0, a_2

t_1, t_0, a_2

$t_1, 0, (máx)$

$a_0, a_0, 4$

f_{echo}

$add: a_1, a_1, -1$

$min: a_0, a_0, 4$

$sdt T_0, 0, (ao)$

t_0, RA

met

f_{echo}

$add: v_1, v_1, 1$

met

f_{echo}

$add: v_0, v_0, 1$

met

tim : f_{reg} delay set não intilizado

lw $T_0, 0, (ao)$

$v_0, 0, 0$

$v_1, 0, 0$

$t_0, 0, 0$

$t_1, a_1, -1$

$a_1, a_1, -1$

$t_0, (ao)$

t_0, t_0, a_2

t_1, t_0, a_2

$t_1, 0, (máx)$

$a_0, a_0, 4$

f_{echo}

$add: a_1, a_1, -1$

$min: a_0, a_0, 4$

$sdt T_0, 0, (ao)$

t_0, RA

met

f_{echo}

$add: v_1, v_1, 1$

met

f_{echo}

$add: v_0, v_0, 1$

met

Arquitetura de Sistemas e Computadores II

Página principal ► As minhas disciplinas ► Licenciaturas ► 2014/2015 - Semestre Impar ► ASC2 ► Práticas ► Exercícios 2^a aula

Exercícios 2^a aula

1.5 [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

- Which processor has the highest performance expressed in instructions per second?
- If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.
- We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

1.6 [20] <§1.6> Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of $1.0E6$ instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster?

- What is the global CPI for each implementation?
- Find the clock cycles required in both cases.

1.7 [15] <§1.6> Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of $1.0E9$ and has an execution time of 1.1 s, while compiler B results in a dynamic instruction count of $1.2E9$ and an execution time of 1.5 s.

- Find the average CPI for each program given that the processor has a clock cycle time of 1 ns.
- Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?
- A new compiler is developed that uses only $6.0E8$ instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

1.9 Assume for arithmetic, load/store, and branch instructions, a processor has CPIs of 1, 12, and 5, respectively. Also assume that on a single processor a program requires the execution of $2.56E9$ arithmetic instructions, $1.28E9$ load/store instructions, and 256 million branch instructions. Assume that each processor has a 2 GHz clock frequency.

Assume that, as the program is parallelized to run over multiple cores, the number of arithmetic and load/store instructions per processor is divided by $0.7 \times p$ (where p is the number of processors) but the number of branch instructions per processor remains the same.

1.9.1 [5] <§1.7> Find the total execution time for this program on 1, 2, 4, and 8 processors, and show the relative speedup of the 2, 4, and 8 processor result relative to the single processor result.

1.9.2 [10] <§§1.6, 1.8> If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors?

1.9.3 [10] <§§1.6, 1.8> To what should the CPI of load/store instructions be reduced in order for a single processor to match the performance of four processors using the original CPI values?

Última alteração: Segunda, 22 Setembro 2014, 01:30

NAVEGAÇÃO

Página principal

Páginas do site

Meu perfil

Disciplina atual

ASC2

Participantes

M

Este é o meu espaço para discutir sobre o que estou estudando e o que estou aprendendo. Aqui posso escrever sobre os meus interesses, desafios e conquistas. Posso também compartilhar artigos, vídeos e outras informações que considero relevantes para o meu aprendizado. Estou sempre aberto a feedback e discussões com outros usuários. Se tiver alguma dúvida ou precisar de ajuda, não hesite em perguntar! Estou aqui para ajudar.

Este é o meu espaço para discutir sobre o que estou estudando e o que estou aprendendo. Aqui posso escrever sobre os meus interesses, desafios e conquistas. Posso também compartilhar artigos, vídeos e outras informações que considero relevantes para o meu aprendizado. Estou sempre aberto a feedback e discussões com outros usuários. Se tiver alguma dúvida ou precisar de ajuda, não hesite em perguntar! Estou aqui para ajudar.

F_{Religio}	1.5
P_1	$3.6 \times 10^9 = 3 \times 10^9$
P_2	$2.56 \times 10^9 = 2.5 \times 10^9$
P_3	$4.64 \times 10^9 = 4.0 \times 10^9$

Processador	Frequência relógio	CPI
P_1	3.6×10^9	1.5
P_2	2.56×10^9	1.0
P_3	4.64×10^9	2.2

15a Qual das processadoras tem a performance maior alta expressa em instruções por segundo?

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

$$\text{CPU Time}_P_1 = \frac{3 \times 10^9}{1.5} = 2 \times 10^9 \rightarrow \text{Execution time} = \frac{\text{mínima} \times \text{CPI}}{\text{Frequência relógio}}$$

$$\text{CPU Time}_P_2 = \frac{2.5 \times 10^9}{1.0} = 2.5 \times 10^9 \rightarrow \text{Tempo de CPU} = \frac{\text{mínimo} \times \text{instruções}}{\text{Tempo de CPU}} = \frac{\text{m}^2 \cdot \text{ciclos}}{\text{Frequência relógio}}$$

$$\text{CPU Time}_P_3 = \frac{4.0 \times 10^9}{2.2} = 1.82 \times 10^9 \rightarrow$$

⇒ P_3 tem a melhor performance

15b

Se cada processador executar um programa em 10 segundos e que o nº de ciclos é o nº de instruções

$$\text{nº ciclos} = (\text{tempo}) \cdot \text{CPU} \times \text{FRE$$

$$\text{nº ciclos}_P_1 = 10 \times 3 \times 10^9 = 3 \times 10^{10}$$

$$\text{nº ciclos}_P_2 = 10 \times 2.5 \times 10^9 = 2.5 \times 10^{10}$$

$$\text{nº ciclos}_P_3 = 10 \times 4 \times 10^9 = 4 \times 10^{10}$$

$$\text{nº instruções} = \frac{\text{tempo de execução} \times \text{F.Religio}}{\text{CPI}} = \frac{10 \times 3 \times 10^9}{1.5} = 2 \times 10^{10}$$

$$\text{nº instruções}_P_2 = \frac{10 \times 2.5 \times 10^9}{1.0} = 2.5 \times 10^{10}$$

$$\text{nº instruções}_P_3 = \frac{10 \times 4 \times 10^9}{2.2} = 1.82 \times 10^{10}$$